



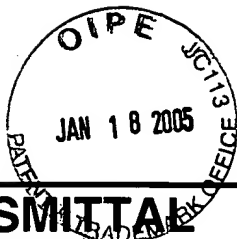
AF 12w

TRANSMITTAL FORM <i>(to be used for all correspondence after initial filing)</i>		Application No.	09/822,090
		Filing Date	March 30, 2001
		First Named Inventor	Guei-Yuan Lueh
		Art Unit	2122
		Examiner Name	Mary J. Steelman
Total Number of Pages in This Submission	22	Attorney Docket Number	42390P10798

ENCLOSURES (check all that apply)		
<input checked="" type="checkbox"/> Fee Transmittal Form <input checked="" type="checkbox"/> Fee Attached <input type="checkbox"/> Amendment / Response <input type="checkbox"/> After Final <input type="checkbox"/> Affidavits/declaration(s) <input type="checkbox"/> Extension of Time Request <input type="checkbox"/> Express Abandonment Request <input type="checkbox"/> Information Disclosure Statement <input type="checkbox"/> PTO/SB/08 <input type="checkbox"/> Certified Copy of Priority Document(s) <input type="checkbox"/> Response to Missing Parts/Incomplete Application <input type="checkbox"/> Basic Filing Fee <input type="checkbox"/> Declaration/POA <input type="checkbox"/> Response to Missing Parts under 37 CFR 1.52 or 1.53	<input type="checkbox"/> Drawing(s) <input type="checkbox"/> Licensing-related Papers <input type="checkbox"/> Petition <input type="checkbox"/> Petition to Convert a Provisional Application <input type="checkbox"/> Power of Attorney, Revocation Change of Correspondence Address <input type="checkbox"/> Terminal Disclaimer <input type="checkbox"/> Request for Refund <input type="checkbox"/> CD, Number of CD(s)	<input type="checkbox"/> After Allowance Communication to Group <input type="checkbox"/> Appeal Communication to Board of Appeals and Interferences <input checked="" type="checkbox"/> Appeal Communication to Group (Appeal Notice, Brief, Reply Brief) <input type="checkbox"/> Proprietary Information <input type="checkbox"/> Status Letter <input type="checkbox"/> Other Enclosure(s) (please identify below): <div style="border: 1px solid black; height: 80px; width: 100%;"></div>
Remarks		

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT	
Firm or Individual name	Thinh V. Nguyen, Reg. No. 42,034 BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
Signature	
Date	January 13, 2005

CERTIFICATE OF MAILING/TRANSMISSION			
I hereby certify that this correspondence is being deposited with the United States Postal Service on the date shown below with sufficient postage as first class mail in an envelope addressed to: Mail Stop Appeal Brief-Patents, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.			
Typed or printed name	Tu T. Nguyen		
Signature		Date	January 13, 2005



FEE TRANSMITTAL for FY 2005

Patent fees are subject to annual revision.

Complete if Known

Application Number	09/822,090
Filing Date	March 30, 2001
First Named Inventor	Guei-Yuan Lueh
Examiner Name	Mary J. Steelman
Art Unit	2122
Attorney Docket No.	42390P10798

☐ Applicant claims small entity status. See 37 CFR 1.27.

TOTAL AMOUNT OF PAYMENT (\$) 500.00

METHOD OF PAYMENT (check all that apply)

☒ Check ☐ Credit card ☐ Money Order ☐ None ☐ Other (please identify): _____

☒ Deposit Account Deposit Account Number: 02-2666 Deposit Account Name: Blakely, Sokoloff, Taylor & Zafman LLP

For the above-identified deposit account, the Director is hereby authorized to: (check all that apply)

☐ Charge fee(s) indicated below

☐ Charge fee(s) indicated below, except for the filing fee

☒ Charge any additional fee(s) or underpayment of fee(s)
under 37 CFR §§ 1.16, 1.17, 1.18 and 1.20.

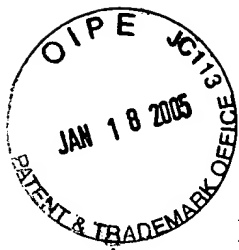
☒ Credit any overpayments

FEE CALCULATION

Large Entity		Small Entity		Fee Description	Fee Paid
Fee Code	Fee (\$)	Fee Code	Fee (\$)		
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet.	
2053	130	2053	130	Non-English specification	
1251	120	2251	60	Extension for reply within first month	
1252	450	2252	225	Extension for reply within second month	
1253	1,020	2253	510	Extension for reply within third month	
1254	1,590	2254	795	Extension for reply within fourth month	
1255	2,160	2255	1,080	Extension for reply within fifth month	
1401	500	2401	250	Notice of Appeal	
1402	500	2402	250	Filing a brief in support of an appeal	500.00
1403	1,000	2403	500	Request for oral hearing	
1451	1,510	2451	1,510	Petition to institute a public use proceeding	
1460	130	2460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
1809	790	1809	395	Filing a submission after final rejection (37 CFR § 1.129(a))	
1810	790	2810	395	For each additional invention to be examined (37 CFR § 1.129(b))	
Other fee (specify) _____					
SUBTOTAL (2)					500.00

SUBMITTED BY

Name (Print/Type)	Thinh V. Nguyen	Registration No. (Attorney/Agent)	42,034	Telephone	(714) 557-3800
Signature		Date	01/13/05		



Docket No.: 042390.P10798

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

<p>In re Application of: Guei_Yuan Lueh Application No.: 09/822,090 Filed: March 30, 2001 For: DEBUGGING SUPPORT USING DYNAMIC RE-COMPILATION</p>	<p>Examiner: Mary J. Steelman Art Group: 2122</p>
---	--

APPEAL BRIEF

Mail Stop Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Sir:

Applicant submits the following Appeal Brief pursuant to 37 C.F.R. § 41.37 for consideration by the Board of Patent Appeals and Interferences. Applicant also submits herewith our check number 31569 in the amount of \$500.00 to cover the cost of filing the opening brief as required by 37 C.F.R. § 1.17(c). Please charge any additional fees or credit any overpayment to our deposit Account No. 02-2666. A duplicate copy of the Fee Transmittal is enclosed for this purpose.

01/19/2005 EABUBAK1 00000049 09822090

01 FC:1402

500.00 OP

TABLE OF CONTENTS

I.	REAL PARTY IN INTEREST	3
II.	RELATED APPEALS AND INTERFERENCES.....	3
III.	STATUS OF CLAIMS	3
IV.	STATUS OF AMENDMENTS	3
V.	SUMMARY OF CLAIMED SUBJECT MATTER	3
VI.	GROUND OF REJECTION TO BE REVIEWED ON APPEAL	5
VII.	ARGUMENTS.....	5
	A. Claims 13-15, 28-30, 35, and 37-39 Use Proper Trademark Names Under 35 U.S.C. §112.:	5
	B. Claims 1, 4-8, 13-16, 19-23, 28-31, 34, and 37-39 Are Not Obvious over <u>Wolczko</u> in View of <u>Angel</u> .:	6
	C. Claims 2-3, 9-12, 17-18, 24-27, 32-34, and 35-36 Are Not Obvious over <u>Wolczko</u> in view of <u>Angel</u> and further in view of <u>Copperman</u> :	9
VIII.	CONCLUSION.....	13
IX.	CLAIMS APPENDIX.....	14

I. REAL PARTY IN INTEREST

The real party in interest is the assignee, Intel Corporation.

II. RELATED APPEALS AND INTERFERENCES

There are no related appeals or interferences known to the appellants, the appellants' legal representative, or assignee, which will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

III. STATUS OF CLAIMS

Claims 1-39 of the present application are pending and remain rejected. The Applicant hereby appeals the rejection of claims 1-39.

IV. STATUS OF AMENDMENTS

The Applicant filed an amendment on September 29, 2004, in response to a Final Office Action issued by the Examiner on August 16, 2004. In response to the September 29, 2004 amendment, the Examiner issued an Advisory Action on November 5, 2004. The Applicant filed a Notice of Appeal from the Advisory Action issued by the Examiner on November 15, 2004.

V. SUMMARY OF CLAIMED SUBJECT MATTER

1. Independent claims 1, 16, and 31:

A debug support 370 includes a stack frame access support 410, a data value access support 420, a control break-point support 430, and a data break-point support 440¹. The data break-point support 440 includes a field access and modifications watch 445 to provide support for field watches². The field access and modification watch 445 has three models or approaches static, semi-static, and dynamic. In the dynamic model, dynamic

¹ Specification, paragraph [0031]; Figure 4.

² Specification, paragraph [0038]; Figure 4.

recompilation is used to insert the instrumentation code when needed³. The dynamic data access support 445 includes a compilation 601 and a recompilation 602⁴.

The recompilation 602 recompiles a function including the byte code sequence when the field watch is activated⁵. The byte code sequence has a field byte code that accesses or modifies a field. The recompilation 602 includes the compilation of byte code 605, the guard 610, a generation of instrumentation code 630, and an insertion instrumentation code 650. The generation 630 generates an instrumentation code corresponding to a field watch of the field⁶. The insertion 650 inserts the instrumentation code to the native code⁷.

2. Dependent claims 2-7, 17-22, and 32-34:

The guard 610 prevents execution of the instrumentation code if the field watch is not activated⁸. The generation 630 includes an execution 640 of a field watch sequence if the field watch is activated. The execution 640 includes saving live global state, executing an event hook function for an event corresponding to the field watch, and restoring the live global state⁹. The live global state corresponds to an active register such as a live scratch register or a floating-point register. In one embodiment, saving the live global state includes pushing the live global state onto a stack, and restoring the saved live global state includes retrieving it from the stack. The execution of the event hook function includes passing an argument corresponding to the field and calling a run-time library function related to the event¹⁰.

3. Dependent claims 8 and 23:

In one embodiment, the insertion 650 inserts the instrumentation code in a stub at the end of the code space of the method¹¹.

4. Dependent claims 9-12, 24-27, and 35-36:

The guard 610 may be implemented by two methods: (1) updating an offset of a jump instruction to the stub when the field watch is activated, and (2) replacing a no-op sequence with a jump instruction to the stub. The use of a jump instruction is simpler

³ Specification, paragraph [0044].

⁴ Specification, paragraph [0052].

⁵ Specification, paragraph [0055]; Figure 6.

⁶ Specification, paragraph [0055].

⁷ Specification, paragraph [0056].

⁸ Specification, paragraph [0054].

⁹ Specification, paragraph [0055]; Figure 6.

¹⁰ Specification, paragraph [0055].

¹¹ Specification, paragraph [0056].

because it merely involves a change of the offset. To clear the field watch, the compiler may either replace the offset of the jump instruction with a zero offset, or replace the jump instruction with a no-op sequence¹².

5. Dependent claims 13-15, 28-30, and 37-39:

In one embodiment, the function is a Java method¹³ In one embodiment, the virtual machine 214 is a Java virtual machine (JVM) implementing the Java Virtual Machine Debug Interface (JVMDI)¹⁴.

VI. GROUND OF REJECTION TO BE REVIEWED ON APPEAL

1. Claims 13-15, 28-30, 35, and 37-39 stand rejected under 35 U.S.C. §112.
2. Claims 1, 4-8, 13-16, 19-23, 28-31, 34, and 37-39 stand rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,078,744 issued to Wolczko et al. ("Wolczko") in view of U.S. Pre Grant Publication 2001/0047510 issued to Angel et al. ("Angel").
3. Claims 2-3, 9-12, 17-18, 24-27, 32-34, 35-36 stand rejected under 35 U.S.C. §103(a) as being unpatentable over Wolczko in view of Angel and further in view of "Poor Man's Watchpoints", by Max Copperman and Jeff Thomas (1995) ("Copperman").

VII. ARGUMENTS

A. Claims 13-15, 28-30, 35, and 37-39 Use Proper Trademark Names Under 35 U.S.C. §112.

In the Final Office Action, the Examiner rejected claims 13-15, 28-30, 35, and 37-39 under 35 U.S.C. §112. The Examiner states that the trademark JAVA identifies the source, i.e., Sun Microsystems, Inc., not goods themselves and suggests Applicant to modify the claims through the use of additional generic terminology such as JAVA programming language method/virtual machine/virtual machine debug interface. However, JAVA is used as an adjective, not as a noun, to definitely characterize a

¹² Specification, paragraph [0054]; Figure 6, blocks 622 and 624.

¹³ Specification, paragraph [0046].

¹⁴ Specification, paragraph [0027].

particular implementation of a function (JAVA method), a field (JAVA field), a virtual machine (JAVA virtual machine), and a debug interface (JAVA virtual machine debug interface). There is no confusion or indefinite regarding the JAVA method, JAVA field, JAVA virtual machine, and the JVM debug interface. Furthermore, JAVA is used in the technical field to indicate a high-level programming language. The use of the JAVA programming language has been so widespread that there is no indefiniteness or confusion.

Furthermore, the rejected claims do recite JAVA through the use of generic terminology as the Examiner suggested. For example, claim 13 recites "a JAVA method", claim 14 recites "a JAVA field" in "a JAVA virtual machine", etc.

Therefore, Applicant believes that the use of JAVA in claims 13-15, 28-30, 35, and 37-39 is proper.

B. Claims 1, 4-8, 13-16, 19-23, 28-31, 34, and 37-39 Are Not Obvious over Wolczko in View of Angel.

In the final Office Action, the Examiner rejected claims 1, 4-8, 13-16, 19-23, 28-31, 34, and 37-39 under 35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 6,078,744 issued to Wolczko et al. ("Wolczko") in view of U.S. Pre Grant Publication 2001/0047510 issued to Angel et al. ("Angel"). Applicant respectfully traverses the rejection and contends that the Examiner has not met the burden of establishing a *prima facie* case of obviousness. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. *MPEP §2143, p. 2100-129 (8th Ed., Rev. 2, May 2004)*. Applicants respectfully contend that there is no suggestion or motivation to combine their teachings, and thus no *prima facie* case of obviousness has been established.

Wolczko discloses a technique to improve compile performance during subsequent compilations of a source program. An optimized compilation process recompiles portions of the source program and generates optimized machine code versions of the heavily used portions (Wolczko, col. 6, lines 51-55).

Angel discloses a byte code instrumentation. A technique to instrument a byte code program includes examining the byte code, selecting portions of the byte code for instrumentation, and instrumenting the portions to provide instrumented byte code (Angel, paragraph [0014]). Memory access instructions are instrumented to detect illegal memory operations at runtime (Angel, paragraph [0091]). In addition, exiting and entering blocks of code where variables become defined and undefined are monitored (Angel, paragraph [0091]).

Angel and Copperman, taken alone or in any combination, does not disclose, suggest, or render obvious (1) re-compiling when a field watch for a field is activated, (2) generating an instrumentation code corresponding to the field watch; and (3) inserting the instrumentation code to the native code. There is no motivation to combine Wolczko and Angel because none of them addresses the problem of recompilation according to a field watch. There is no teaching or suggestion that a field byte code accessing or modifying the field is present. Wolczko, read as a whole, does not suggest the desirability of generating an instrumentation code corresponding to the field watch.

The Examiner states that Wolczko discloses recompiling byte code into native code (Final Office Action, page 3). However, Wolczko does not disclose recompiling a function when a field watch for a field is activated. Wolczko merely discloses recompiling a portion of a program that is heavily used (Wolczko, col. 6, lines 54-55). The condition for recompilation is the frequency of usage, not the activation of a field watch.

The Examiner further states that Angel discloses, activating field watch by monitoring memory access instructions and variables of a program (Final Office Action, page 4). Applicant respectfully disagrees. Monitoring memory access instructions and/or variables is not the same as activating a field watch of a field. A field watch sequence may include instruction sequence to spill the mimic stack operands, which are live at the field access point, to their canonical spill locations (See, for example, Specification, page 14, paragraph [0047]).

In the Final Office Action, the Examiner responds to Applicant's arguments. Applicant contends that the Examiner's responses failed to overcome Applicant's arguments. The Examiner states that the Wolczko reference suggests recompiling altered code. Applicant respectfully disagrees. Wolczko merely discloses recompiling portions of a source program which is an interpreted code while the code is being executed (Wolczko,

col. 6, lines 52-55). Since the code is being executed, it is impossible to alter the code. Wolczko discloses optimizing a compiler by not re-creating the related computationally expressive information.

The Examiner further states that “field watch” is a broad term. Applicant respectfully disagrees. Claim must be interpreted according to the Specification. See Renishaw PLC v. Marposs Societa’ per Azioni, 158 F.3d 1248, 48 USPQ 2d at 1117 (Fed. Cir. 1998). A field is a JAVA variable that is defined in a JAVA object. Field access events are generated when the field specified is about to be accessed. Field accesses from JAVA Language Code or from JNI are watched. (See Specification, page 12, paragraph [0039].

The Examiner further states that the “field” could be the means of determination as whether a node in the Intermediate Representation (IR) tree is a “node of interest” (Final Office Action, page 12). Applicant respectfully disagrees. Angel merely discloses that a node of interest includes any node that is to be instrumented or which indicates that instrumentation is appropriate (Angel, paragraph [0089]). This is not related to a field watch.

The Examiner further states that Angel discloses recompilation in that the compiler may continue the compile process (Final Office Action, page 12). Applicant respectfully disagrees. Angel merely discloses that once the instrumented IR data element is provided, then the compiler may continue the compile process by accessing the instrumented IR data element to provide the object code (Angel, paragraph [0123]). Continuing the compile process is not the same as recompiling. Since Angel does not disclose or suggest recompiling, and neither Wolczko nor Angel discloses or suggests using a field watch, the combination of Wolczko and Angel is improper.

In the Advisory Action dated November 5, 2004, the Examiner, citing paragraphs [0125] and [0127] in Angel, states that Angel provided references that determine if a field is to be “watched”, then instrumentation code is generated and inserted. Applicant respectfully disagrees for the following reasons.

For ease of reference, the cited paragraphs in Angel are copied below.

*{0125} Other embodiments also exist. Described
below are methods of automatically editing the executable
byte code representation of a computer program or other*

methods for generating instrumented byte code. In one embodiment, the byte code is altered by the addition of new instructions and/or the deletion or modification of existing instructions.

[0127] One objective of the instrumentation process is to alter the program to facilitate the gathering of diagnostic and statistical information on the program when it is executed; i.e., dynamic analysis. This allows the program's internal state to be monitored for variety of purposes. These purposes include, but are not limited to: diagnosing error conditions that occur at run time, creating a record of the inner details of program execution, measuring program execution to provide code coverage analysis and performance profiling, or providing additional run time error or exception handling.

As shown in the above, there is no reference of a field watch. The reference that “[t]his allows the program’s internal state to be monitored for variety of purposes” in paragraph [0127] merely refers to the program’s internal state, not a field watch. As discussed above, a memory variable access is not a field watch. A field is a Java variable that is defined in a Java object. Field access events are generated when the field specified is about to be accessed. Field accesses from Java Language Code or from JNI are watched (See Specification, page 12, paragraph [0039].

Therefore, Applicant believes that independent claims 1, 16, 31 and their respective dependent claims are distinguishable over the cited prior art references.

C. Claims 2-3, 9-12, 17-18, 24-27, 32-34, and 35-36 Are Not Obvious over Wolczko in view of Angel and further in view of Copperman

In the final Office Action, the Examiner rejected claims 2-3, 9-12, 17-18, 24-27, 32-34, and 35-36 under 35 U.S.C. §103(a) as being unpatentable over Wolczko in view of Angel and further in view of “Poor Man’s Watchpoints”, by Max Copperman and Jeff Thomas (1995) (“Copperman”). Applicant respectfully traverses the rejection and

contends that the Examiner has not met the burden of establishing a prima facie case of obviousness.

Wolczko discloses a technique to improve compile performance during subsequent compilations of a source program, as discussed above. Angel discloses a byte code instrumentation, as discussed above.

Copperman discloses a technique to implement watch points using code patching. When the user sets a watch point, the debugger sets the register \$fp to point to a register save area in the debuggee's static data space. When no watch points are set, the first instruction in the patch branches around the rest of the patch if \$fp contains (Copperman, page 38, third paragraph under section "The Debuggee"). Watchpoints in Copperman allow the user to get control at the point that the location is written to, that is, at the assignment through the pointer, allowing the user to identify the errant pointer (Copperman, first paragraph under the section "Introduction"). Since Copperman's watchpoint refers to a memory location referenced by a pointer, it is not the same as a field watch.

Wolczko, Angel and Copperman, taken alone or in any combination, does not disclose, suggest, or render obvious (1) re-compiling when a field watch for a field is activated, (2) generating an instrumentation code corresponding to the field watch; (3) inserting the instrumentation code to the native code; (4) guarding execution of the instrumentation code if the field watch is not activated; and (5) executing a field watch sequence if the field watch is activated. There is no motivation to combine Wolczko, Angel and Copperman because none of them addresses the problem of recompilation according to a field watch. There is no teaching or suggestion that a field byte code accessing or modifying the field is present. Wolczko, read as a whole, does not suggest the desirability of generating an instrumentation code corresponding to the field watch, or guarding execution of the instrumentation code if the field watch is not activated, and executing a field watch sequence if the field watch is activated.

The Examiner states that Copperman discloses guarding execution of the instrumentation code if the field watch is not activated, and executing a field watch sequence if the field watch is activated, citing page 38, third paragraph of the section "The Debuggee", and page 40, second paragraph of the section "Maintaining The Watch Table", and page 41, third paragraph. Applicant respectfully disagrees. Disabling or clearing a

command is not the same as activating or clearing a field watch. A command is not a field watch. Furthermore, as argued above, a watchpoint is not the same as a field watch. Even if enabling/ clearing a command is the same as enabling/ clearing a field watch, Copperman does not disclose or suggest guarding execution of the instrumentation code if the field watch is not activated, or executing a field watch sequence if the field watch is activated. Copperman merely discloses that when a watchpoint command is entered or enabled, the address range is appended to the debugger's watch table, or when a command is disabled or canceled, the last range in the table is copied over the range that is no longer being watched (Copperman, page 40, second paragraph under section "Maintaining the Watch Table"). Clearly, appending the address range to the watch table has nothing to do with "executing a field watch sequence", and copying the last range in the table is not the same as "guarding execution of the instrumentation code".

Finally, as argued above in the independent claims 1, 16, and 31, Angel and Copperman, taken alone or in combination, do not disclose or render obvious: (1) re-compiling when a field watch for a field is activated, (2) generating an instrumentation code corresponding to the field watch; and (3) inserting the instrumentation code to the native code. Accordingly, Wolczko, Angel and Copperman, taken alone or in combination, do not disclose or render obvious guarding execution of the instrumentation code if the field watch is not activated, and executing a field watch sequence if the field watch is activated.

In summary, the Examiner failed to establish a prima facie case of obviousness and failed to show there is teaching, suggestion or motivation to combine the references. "When determining the patentability of a claimed invention which combined two known elements, 'the question is whether there is something in the prior art as a whole suggest the desirability, and thus the obviousness, of making the combination.'" In re Beattie, Lindemann Maschinenfabrik GmbH v. American Hoist & Derrick Co., 730 F.2d 1452, 1462, 221 USPQ (BNA) 481, 488 (Fed. Cir. 1984). To defeat patentability based on obviousness, the suggestion to make the new product having the claimed characteristics must come from the prior art, not from the hindsight knowledge of the invention. Interconnect Planning Corp. v. Feil, 744 F.2d 1132, 1143, 227 USPQ (BNA) 543, 551 (Fed. Cir. 1985). To prevent the use of hindsight based on the invention to defeat patentability of the invention, this court requires the Examiner to show a motivation to

combine the references that create the case of obviousness. In other words, the Examiner must show reasons that a skilled artisan, confronted with the same problems as the inventor and with no knowledge of the claimed invention, would select the prior elements from the cited prior references for combination in the manner claimed. In re Rouffet, 149 F.3d 1350 (Fed. Cir. 1996), 47 USPQ 2d (BNA) 1453. "To support the conclusion that the claimed invention is directed to obvious subject matter, either the references must expressly or implicitly suggest the claimed invention or the Examiner must present a convincing line of reasoning as to why the artisan would have found the claimed invention to have been obvious in light of the teachings of the references." Ex parte Clapp, 227 USPQ 972, 973. (Bd.Pat.App.&Inter. 1985). The mere fact that references can be combined or modified does not render the resultant combination obvious unless the prior art also suggests the desirability of the combination. In re Mills, 916 F.2d 680, 16 USPQ2d 1430 (Fed. Cir. 1990). Furthermore, although a prior art device "may be capable of being modified to run the way the apparatus is claimed, there must be a suggestion or motivation in the reference to do so." In re Mills 916 F.2d at 682, 16 USPQ2d at 1432; In re Fitch, 972 F.2d 1260, 23 USPQ2d 1780 (Fed. Cir. 1992).

In the present invention, the cited references do not expressly or implicitly suggest any one of the above elements. In addition, the Examiner failed to present a convincing line of reasoning as to why a combination of Wolczko, Angel, or a combination of Wolczko, Angel and Copperman is an obvious application of re-compilation using the field watch in debugger support.

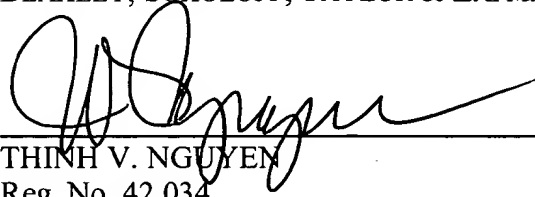
Therefore, Applicant believes that independent claims 1, 16, 31 and their respective dependent claims are distinguishable over the cited prior art references.

VIII. CONCLUSION

Applicant respectfully requests that the Board enter a decision overturning the Examiner's rejection of all pending claims, and holding that the claims are neither anticipated nor rendered obvious by the prior art.

Respectfully submitted,

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP

A handwritten signature in black ink, appearing to read 'Thinh V. Nguyen', is written over a horizontal line.

THINH V. NGUYEN
Reg. No. 42,034

Dated: January 12, 2005

12400 Wilshire Blvd., 7th Floor
Los Angeles, CA 90025-1026
(714) 557-3800

IX. CLAIMS APPENDIX

The claims of the present application which are involved in this appeal are as follows:

1. (original) A method comprising:
re-compiling a function when a field watch for a field is activated, the function including a byte code sequence having a field byte code that accesses or modifies the field, the recompiled function providing a native code and occupying a code space;
generating an instrumentation code corresponding to the field watch of the field;
and
inserting the instrumentation code to the native code.
2. (original) The method of claim 1 further comprising:
guarding execution of the instrumentation code if the field watch is not activated.
3. (original) The method of claim 1 wherein generating the instrumentation code comprises:
executing a field watch sequence if the field watch is activated.
4. (previously presented) The method of claim 3 wherein executing the field watch sequence comprises:
saving live global state, the live global state corresponding to an active register;
executing an event hook function for an event corresponding to the field watch; and
restoring the live global state.
5. (original) The method of claim 4 wherein saving the live global state comprises:
pushing the live global state onto a stack.
6. (original) The method of claim 4 wherein executing the event hook function comprises:
passing an argument corresponding to the field; and

calling a run-time library function related to the event.

7. (original) The method of claim 5 wherein restoring the live global state comprises:

retrieving the live global state from the stack.

8. (original) The method of claim 1 wherein inserting the instrumentation code comprises:

inserting the instrumentation code in a stub at end of the code space.

9. (previously presented) The method of claim 8 wherein guarding execution of the instrumentation code comprises:

updating an offset of a jump instruction to the stub when the field watch is activated.

10. (previously presented) The method of claim 8 wherein guarding execution of the instrumentation code comprises:

replacing a no-op sequence with a jump instruction to the stub.

11. (original) The method of claim 9 further comprising:

clearing the field watch by replacing the offset with a zero offset.

12. (original) The method of claim 10 further comprising:

clearing the field watch by replacing the jump instruction with the no-op sequence.

13. (previously presented) The method of claim 1 wherein the function is a JAVA method.

14. (previously presented) The method of claim 1 wherein the field is a JAVA field in a JAVA virtual machine.

15. (previously presented) The method of claim 4 wherein the event hook function is compatible with a JAVA Virtual Machine Debug Interface (JVMDI).

16. (original) A computer program product comprising:
a machine useable medium having computer program code embedded therein, the computer program product having:

computer readable program code to re-compile a function when a field watch for a field is activated, the function including a byte code sequence having a field byte code that accesses or modifies the field, the recompiled function providing a native code and occupying a code space,

computer readable program code to generate an instrumentation code corresponding to the field watch of the field, and

computer readable program code to insert the instrumentation code to the native code.

17. (original) The computer program product of claim 16 further comprising:
computer readable program code to guard execution of the instrumentation code if the field watch is not activated.

18. (original) The computer program product of claim 16 wherein the computer readable program code to generate the instrumentation code comprises:

computer readable program code to execute a field watch sequence if the field watch is activated.

19. (previously presented) The computer program product of claim 18 wherein the computer readable program code to execute the field watch sequence comprises:

computer readable program code to save live global state, the live global state corresponding to an active register;

computer readable program code to execute an event hook function for an event corresponding to the field watch; and

computer readable program code to restore the live global state.

20. (original) The computer program product of claim 19 wherein the computer readable program code to save the live global state comprises:

computer readable program code to push the live global state onto a stack.

21. (original) The computer program product of claim 19 wherein the computer readable program code to execute the event hook function comprises:

computer readable program code to pass an argument corresponding to the field;
and

computer readable program code to call a run-time library function related to the event.

22. (original) The computer program product of claim 20 wherein the computer readable program code to restore the live global state comprises:

computer readable program code to retrieve the live global state from the stack.

23. (original) The computer program product of claim 16 wherein the computer readable program code to insert the instrumentation code comprises:

computer readable program code to insert the instrumentation code in a stub at end of the code space.

24. (previously presented) The computer program product of claim 23 wherein the computer readable program code to guard execution of the instrumentation code comprises:

computer readable program code to update an offset of a jump instruction to the stub when the field watch is activated.

25. (previously presented) The computer program product of claim 23 wherein the computer readable program code to guard execution of the instrumentation code comprises:

computer readable program code to replace a no-op sequence with a jump instruction to the stub.

26. (original) The computer program product of claim 24 further comprising:
computer readable program code to clear the field watch by replacing the offset with a zero offset.

27. (original) The computer program product of claim 25 further comprising:

computer readable program code to clear the field watch by replacing the jump instruction with the no-op sequence.

28. (previously presented) The computer program product of claim 16 wherein the function is a JAVA method.

29. (previously presented) The computer program product of claim 16 wherein the field is a JAVA field in a JAVA virtual machine.

30. (previously presented) The computer program product of claim 19 wherein the event hook function is compatible with a JAVA Virtual Machine Debug Interface (JVMDI).

31. (original) A system comprising:
a processor;
a memory coupled to the processor to store instruction code, the instruction code, when executed by the processor, causing the processor to:
re-compile a function when a field watch for a field is activated, the function including a byte code sequence having a field byte code that accesses or modifies the field, the re-compiled function providing a native code and occupying a code space,
generate an instrumentation code corresponding to the field watch of the field, and
insert the instrumentation code to the native code.

32. (original) The system of claim 31 the instruction code further causing the processor to:
guard execution of the instrumentation code if the field watch is not activated.

33. (original) The system of claim 31 wherein the instruction code causing the processor to generate the instrumentation code causes the processor to:
execute a field watch sequence if the field watch is activated.

34. (previously presented) The system of claim 33 wherein the instruction code causing the processor to execute the field watch sequence causes the processor to:
save live global state, the live global state corresponding to an active register;
execute an event hook function for an event corresponding to the field watch; and
restore the live global state.

35. (previously presented) The system of claim 32 wherein the instruction code causing the processor to guard execution of the instrumentation code causes the processor to:
update an offset of a jump instruction to a stub when the field watch is activated.

36. (previously presented) The system of claim 32 wherein the instruction code causing the processor to guard execution of the instrumentation code causes the processor to:
replace a no-op sequence with a jump instruction to a stub.

37. (previously presented) The system of claim 31 wherein the function is a
JAVA method.

38. (previously presented) The system of claim 31 wherein the field is a JAVA
field in a JAVA virtual machine.

39. (previously presented) The system of claim 34 wherein the event hook
function is compatible with a JAVA Virtual Machine Debug Interface (JVMDI).